hubbells.game <- function(s, a, nc, dc, nsim, details=F) # s = número de espécies inicial, a = número inicial de indivíduos por espécie, nc = número de ciclos, dc = número de mortes por ciclo, nsim = número de simulações

{

# Verificando se há argumentos faltantes

if(missing(s)) stop("É obrigatória a definição de um valor para o número inicial de espécies (s).") # caso o argumento "s" esteja faltante, a função é interrompida e é enviada uma mensagem de erro

if(missing(a)) stop("É obrigatória a definição de um valor para a abundância inicial das espécies (a).") # caso o argumento "a" esteja faltante, a função é interrompida e é enviada uma mensagem de erro

if(missing(nc)){ # caso o argumento "nc" esteja faltante

nc=0 # será utilizado nc = 0

warning("Não foi definido um valor para o número de ciclos a serem rodados (nc). Logo, foi utilizado nc = 0.") # e uma mensagem de aviso será retornada

}

if(missing(dc)){ # caso o argumento "dc" esteja faltante

dc=1 # será utilizado dc = 1

warning("Não foi definido um valor para o número de mortes por ciclo (dc). Logo, foi utilizado dc = 1, presente no modelo neutro clássico de Hubbell.") # e uma mensagem de aviso será retornada

}

if(missing(nsim)){ # caso o argumento "nsim" esteja faltante

nsim=1 # será utilizado nsim = 1

warning("Não foi definido um valor para o número de simulações a serem rodadas (nsim). Logo, foi utilizado nsim = 1.") # e uma mensagem de aviso será retornada

}

# Verificando se os valores determinados pelo usuário para os argumentos são lógicos

if(s<=0) stop("O número inicial de espécies deve ser maior que zero.") # caso o valor do argumento "s" seja menor ou igual a zero, a função é interrompida e é enviada uma mensagem de erro

if(a<=0) stop("A abundância inicial das espécies deve ser maior que zero.") # caso o valor do argumento "a" seja menor ou igual a zero, a função é interrompida e é enviada uma mensagem de erro

if(nc==0){ # caso o valor do argumento "nc" seja igual a zero

warning("O número de ciclos (nc) definido foi zero.") # é enviada uma mensagem de aviso

}

if(nc<0) stop("O número de ciclos (nc) definido deve ser maior ou igual a zero.") # caso o valor do argumento "nc" seja menor que zero, a função é interrompida e é enviada uma mensagem de erro

if(dc==0 & nc>0) stop("O número de mortes por ciclo (dc) deve ser maior que zero.") # caso o valor do argumento "dc" seja zero, a função é interrompida e é enviada uma mensagem de erro

if(dc<0) stop("O número de mortes por ciclo (dc) definido deve ser maior ou igual a zero.") # caso o valor do argumento "dc" seja menor que zero, a função é interrompida e é enviada uma mensagem de erro

if(nsim<=0) stop("O número de simulações deve ser maior que zero.") # caso o valor do argumento "nsim" seja menor ou igual a zero, a função é interrompida e é enviada uma mensagem de erro

if(dc>=(s\*a)) stop ("O número de mortes por ciclo deve ser menor do que o número de indivíduos total da comunidade.") # caso o número de de mortes por ciclo determinado seja maior ou igual ao número de indivíduos presentes na comunidade, a função é interrompida e é enviada uma mensagem de erro

# Após todas as verificações: Cálculo inicial

n <- s\*a # número total de indivíduos na comunidade (valor fixo)

# Criando array para guardar os resultados

resultado.array <- array(data=,dim=c((nc+1),n,nsim),dimnames=list(paste("ciclo",0:(nc),sep=" "),paste("ind",1:n,sep=" "),paste("simulação",1:(nsim),sep=" "))) # cria array cuja primeira dimensão é composta por linhas referentes ao número de ciclos rodados, cuja segunda dimensão é composta por colunas referentes aos indivíduos presentes na comunidade (as linhas e colunas formam matrizes) e cuja terceira dimensão é composta por nsim matrizes, referentes ao número de simulações rodadas.

# Condições iniciais

vetor.esp <- rep(1:s, each=a) # vetor com as espécies de todos os indivíduos no ciclo 0 (retiradas das condições iniciais determinadas pelo usuário)

# Iniciando: caso o número de ciclos seja zero

if(nc==0)

{

## Guardando as condições iniciais na primeira linha das matrizes do array

resultado.array[1,,] <- vetor.esp # o vetor com a identidade dos indivíduos (espécies a que pertencem) é guardado na linha 1 de todas as colunas em todas as matrizes do array

## Alterando parâmetros para construção de gráficos

par(mfrow=c(1,2)) # altera os parâmetros gráficos para plotarmos 2 gráficos na mesma janela, um ao lado do outro

## Criando o gráfico de distribuição de abundâncias

abund.list <- list() # cria lista vazia para guardar a distribuição de abundância final de cada simulação

for (i in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado pelo usuário (nsim)

abund.list[[i]] <- table(table(resultado.array[(nc+1),,i])) # guarda, na lista criada, o valor de distribuição de abundância final de cada simulação

}

for(i in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado pelo usuário (nsim)

names(abund.list[[i]]) <- jitter(as.numeric(names(abund.list[[i]]))) # substitui os valores de abundância calculados por valores acrescidos de um ruído, para evitar sobreposições ao plotarmos o gráfico de distribuição de abundância

}

plot(abund.list[[1]],lty=2,xlab="Abundância Final",ylab="Frequência de Espécies",main="Distribuição de Abundância",col=1,ylim=c((min(sapply(abund.list,min))),(max(sapply(abund.list,max)))),xlim=c((round(min(sapply(X=abund.list,FUN=function(X){min(as.numeric(names(X)))})))-1),(round(max(sapply(X=abund.list,FUN=function(X){max(as.numeric(names(X)))})))+1)),axes=F) # plota a distribuição de abundância final da primeira simulação, presente na primeira posição da lista criada

if(nsim>1){ # caso o número de simulações seja maior que 1, devemos continuar plotando

for(i in 2:nsim){ # cria loop com contador de 2 até o número de simulações determinado pelo usuário

lines(abund.list[[i]],lty=2,col=i) # plota as distribuições de abundância finais das demais simulações na mesma janela gráfica. Cada simulação tem uma cor própria.

}

}

axis(1,at=c(round((min(sapply(X=abund.list,FUN=function(X){min(as.numeric(names(X)))})))):round((max(sapply(X=abund.list,FUN=function(X){max(as.numeric(names(X)))}))))),labels=c(round((min(sapply(X=abund.list,FUN=function(X){min(as.numeric(names(X)))})))):round((max(sapply(X=abund.list,FUN=function(X){max(as.numeric(names(X)))})))))) # plota o eixo x

axis(2,at=c((min(sapply(abund.list,min))):(max(sapply(abund.list,max)))),labels=c((min(sapply(abund.list,min))):(max(sapply(abund.list,max))))) # plota o eixo y

## Criando o gráfico de riqueza x nc

s.final <- c() # cria um vetor vazio para guardar a riqueza final de cada simulação

for(i in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado

s.final[i] <- length(unique(resultado.array[(nc+1),,i])) # extrai do array a riqueza final de cada simulação e guarda no vetor criado

}

s.final.med <- mean(s.final) # guarda em um objeto a média da riqueza final de espécies da comunidade

plot(s.final[1],type="p",main="Variação da Riqueza",ylab="Riqueza",xlab="Ciclo",col=1,ylim=c(s,s),axes=F) # plota a riqueza em função do número de ciclos rodados. Não há necessidade de plotar a riqueza final de todas as simulações, já que ela é a mesma para todas.

if(nsim>1){ # caso o número de simulações seja maior que 1, devemos continuar plotando

for(i in 2:nsim){ # cria loop com contador de 2 até o número de simulações determinado pelo usuário

points(jitter(s.final[i]),lty=2,col=i) # plota as distribuições de abundância finais das demais simulações na mesma janela gráfica (com adição de ruído, para evitar sobreposição gráfica dos pontos). Cada simulação tem uma cor própria.

}

}

axis(side=1,at=c(0:2),labels=c(-1:1)) # plota o eixo x do gráfico

axis(side=2,at=c(0,s.final.med,(2\*s.final.med)),labels=c("",s.final.med,"")) # plota o eixo y do gráfico

axis(side=3,at=c(0,2),labels=c("","")) # plota a borda superior do gráfico

axis(side=4,at=c(0,(2\*s.final.med)),labels=c("","")) # plota a borda direita do gráfico

## Retornando aos parâmetros usuais

par(mfrow=c(1,1)) # retorna os parâmetros gráficos para o default do R

## Mensagem de aviso acerca dos resíduos gerados para construção do gráfico

warning("A fim de evitar sobreposição de linhas ou pontos com mesmo valor de x e/ou y nos gráficos, estes foram plotados a partir de um resíduo aleatório gerado para cada valor original de x ou y.") # retorna mensagem de aviso acerca dos resíduos associados aos valores de x ou y nos gráficos

## Outputs

### Opção sem retorno do array

if(details==F){ # não retorna o array "resultado.array"

cat("Média da riqueza final\n") # retorna no console a frase "Média do número de espécies final"

return(s.final.med) # retorna a média da riqueza final

}

### Opção com retorno do array

if(details==T){ # retorna o array "resultado.array"

resulta=list("Média da riqueza final"=s.final.med,"Indentidade dos indivíduos ao final de cada ciclo"=resultado.array) # cria uma lista, chamada "resulta", com a média da riqueza final e com o array "resultado.array"

return(resulta) # retorna a lista "resulta"

}

}

# Iniciando: caso o número de ciclos seja maior que zero

else

{

## Guardando as condições iniciais na primeira linha das matrizes do array

resultado.array[1,,] <- vetor.esp

## Rodando os ciclos

for(k in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado pelo usuário (nsim)

for(i in (2:(nc+1))){ # cria loop com contador de 2 até o número de ciclos determinado pelo usuário (nc) + 1

## Morte

ind.morto <- sample(x=n,size=dc,replace=F) # seleção de dc indivíduos que irão morrer (sem reposição) a partir de n indivíduos, com igual probabilidade

vetor.esp[c(ind.morto)] <- NA # morte dos indivíduos selecionados, atribuindo NA a suas identidades

## Nascimento

esp.ind.novo <- sample(x=na.omit(vetor.esp),size=dc,replace=T) # seleção da espécie dos indivíduos novos a partir das espécies dos indivíduos restantes (com reposição)

vetor.esp[is.na(vetor.esp)] <- esp.ind.novo #substituição das espécies dos indivíduos mortos pelas espécies dos indivíduos novos

resultado.array[i,,k] <- vetor.esp # guarda o resultado nas linhas de 2 até (nc+1) (na linha 1, já estão gravadas as condições iniciais) das matrizes do array

}

}

## Alterando parâmetros para construção de gráficos

par(mfrow=c(1,2)) # altera os parâmetros gráficos para plotarmos 2 gráficos na mesma janela, um ao lado do outro

## Criando o gráfico de distribuição de abundâncias

abund.list <- list() # cria lista vazia para guardar a distribuição de abundância final de cada simulação

for (i in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado pelo usuário

abund.list[[i]] <- table(table(resultado.array[(nc+1),,i])) # guarda, na lista criada, o valor de distribuição de abundância final de cada simulação

}

for(i in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado pelo usuário (nsim)

names(abund.list[[i]]) <- jitter(as.numeric(names(abund.list[[i]]))) # substitui os valores de abundância calculados por valores acrescidos de um ruído, para evitar sobreposições ao plotarmos o gráfico de distribuição de abundância

}

plot(abund.list[[1]],lty=2,xlab="Abundância Final",ylab="Frequência de Espécies",main="Distribuição de Abundância",col=1,ylim=c((min(sapply(abund.list,min))),(max(sapply(abund.list,max)))),xlim=c((round(min(sapply(X=abund.list,FUN=function(X){min(as.numeric(names(X)))})))-1),(round(max(sapply(X=abund.list,FUN=function(X){max(as.numeric(names(X)))})))+1)),axes=F) # plota a distribuição de abundância final da primeira simulação, presente na primeira posição da lista criada

if(nsim>1){ # caso o número de simulações seja maior que 1, devemos continuar plotando

for(i in 2:nsim){ # cria loop com contador de 2 até o número de simulações determinado pelo usuário

lines(abund.list[[i]],lty=2,col=i) # plota as distribuições de abundância finais das demais simulações na mesma janela gráfica. Cada simulação tem uma cor própria.

}

}

axis(1,at=c(round((min(sapply(X=abund.list,FUN=function(X){min(as.numeric(names(X)))})))):round((max(sapply(X=abund.list,FUN=function(X){max(as.numeric(names(X)))}))))),labels=c(round((min(sapply(X=abund.list,FUN=function(X){min(as.numeric(names(X)))})))):round((max(sapply(X=abund.list,FUN=function(X){max(as.numeric(names(X)))})))))) # plota eixo x

axis(2,at=c(1:(max(sapply(abund.list,max)))),labels=c(1:(max(sapply(abund.list,max))))) # plota eixo y

## Criando o gráfico de riqueza x nc

s.final <- c() # cria um vetor vazio para guardar a riqueza final de cada simulação

for(i in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado (nsim)

s.final[i] <- length(unique(resultado.array[(nc+1),,i])) # extrai do array a riqueza final de cada simulação e guarda no vetor criado

}

s.final.med <- mean(s.final) # guarda em um objeto a média da riqueza final de espécies da comunidade

s.final.mat <- matrix(data=,nrow=(nc+1),ncol=nsim) # cria uma matriz vazia com nc+1 linhas e nsim colunas para guardar as riquezas de todas as simulações a cada ciclo rodado

for(k in 1:nsim){ # cria loop com contador de 1 até o número de simulações determinado pelo usuário (nsim)

for(j in 1:(nc+1)) { # cria loop com contador de 1 até o número de ciclos determinado pelo usuário mais 1 (nc+1)

s.final.mat[j,k] <- (length(unique(resultado.array[j,,k]))) # extrai do array a riqueza da comunidade a cada ciclo rodado em todas as simulações e guarda na matriz criada

}

}

matplot((s.final.mat),type="l",main="Variação da Riqueza",ylab="Riqueza",xlab="Ciclo",col=1:nsim,lty=2,xlim=c(1,(nc+1)),ylim=c(1,s),axes=F) # plota a riqueza em função do número de ciclos rodados para todas as simulações

axis(1,at=c((-(nc+1)):(2\*(nc+1))),labels=c((-(nc+2)):((2\*(nc+1))-1))) # plota eixo x do gráfico

axis(2,at=c(-1:(2\*s)),labels=c(-1:(2\*s))) # plota eixo y do gráfico

axis(3,at=c((-(nc+1)),(2\*(nc+1))),labels=c("","")) # plota borda superior do gráfico

axis(4,at=c(-1,2\*s),labels=c("","")) # plota borda direita do gráfico

## Retornando aos parâmetros usuais

par(mfrow=c(1,1)) # retorna os parâmetros gráficos para o default do R

## Mensagem de aviso acerca dos resíduos gerados para construção do gráfico

warning("A fim de evitar sobreposição de linhas ou pontos com mesmo valor de x e/ou y nos gráficos, estes foram plotados a partir de um resíduo aleatório gerado para cada valor original de x ou y.") # retorna mensagem de aviso acerca dos resíduos associados aos valores de x ou y nos gráficos

## Outputs

### Opção com retorno do array

if(details==T){ # retorna o array "resultado.array"

resulta=list("Média da riqueza final"=s.final.med,"Indentidade dos indivíduos ao final de cada ciclo"=resultado.array) # cria uma lista, chamada "resulta", com a média da riqueza final e com o array "resultado.array"

return(resulta) # retorna a lista "resulta"

}

### Opção sem retorno do array

if(details==F){ # não retorna o array "resultado.array"

cat("Média da riqueza final\n") # retorna no console a frase "Média da riqueza final"

return(s.final.med) # retorna a média da riqueza final

}

}

} # Fim da função hubbells.game()

## FALTA FAZER:

# colocar ou não o jitter do graf de riqueza qd nc!=0 ? pensar em jeito melhor